REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT DATE	3. DATES COVERED *(From - To)*
15 - 07 - 2002	Annual Technical Progress	09 July 2001-30 June 2002

4. TITLE AND SUBTITLE
Language-based Security for Malicious Mobile Code

5a. CONTRACT NUMBER

5b. GRANT NUMBER
N00014-01-1-0968

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Fred B. Schneider,
Dexter Kozen,
Greg Morrisett and
Andrew Myers

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Cornell University
Ithaca, NY 14853

8. PERFORMING ORGANIZATION REPORT NUMBER
39545

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Office of Naval Research
Ballston Centre Tower ONE
800 North Quincy Street
Arlington, VA 22217-5660

10. SPONSOR/MONITOR'S ACRONYM(S)
ONR

11. SPONSORING/MONITORING AGENCY REPORT NUMBER

12. DISTRIBUTION AVAILABILITY STATEMENT
Unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Report summarizes progress over the past year in developing language-based technologies for defending software systems against attacks from mobile code and system extensions.

20020718 080

15. SUBJECT TERMS
In-lined reference monitors, proof carrying code, end-to-end security, information flow enforcement

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPONE NUMBER *(Include area code)*
U	U	U	UU		

Language-based Security for Malicious Mobile Code

N00014-01-1-0968

Annual Report
09 July 2001 – 30 June 2002

Fred B. Schneider (Principal Investigator)
Dexter Kozen, Greg Morrisett, and Andrew Myers (Co-Investigators)

Department of Computer Science
Cornell University
Ithaca, New York 14853

Overview

Mobile code provides a convenient, efficient, and economical way to extend the functionality and improve the performance of networked computing systems. It is an approach that has been widely embraced, as evidenced by today's operating systems, web browsers, and applications with their support for "plug-and-play", Javascript, downloaded helper applications, and executable attachments. Yet today's security architectures provide poor protection from faulty, much less from malicious, extensions. Our information systems are thus increasingly susceptible to attacks—attacks that can have devastating consequences.

This project is investigating programming language technology—program analysis and program rewriting—for defending software systems against attacks from mobile code and system extensions. The approach promises to support a wide range of flexible, fine-grained access-control and information-flow policies. Only a small trusted computing base seems to be required. And the run-time costs of enforcement should be low.

Our progress over the past year is summarized below. Details can be found in the publications whose citations are given following all the summaries. A list of DoD interactions and technology transitions appears at the end of the report.

In-lined Reference Monitors

In-lined reference monitors are a new approach to implementing traditional reference monitors. A desired end-to-end security policy is formulated using a high-level declarative policy language., and then a rewriting tool is used to automatically rewrite untrusted code into code that respects the policy. The rewriting tool works by inserting extra state and dynamic checks into the untrusted code so that the code becomes self-monitoring.

Over the past year, we made progress towards understanding issues associated with the deployment of IRMs in a production operating system. A set of kernel modifications was developed to support a prototype IRM rewriter in Microsoft's Windows. This work seems to suggest the need for mechanism to identify which policy is applied to any given executable and for mechanism to manage multiple executables (each enforcing a different policy). For example, .NET caches dll's (executables), and the architecture for how that cache is managed needs to work differently when the same dll could have been rewritten in multiple ways (to enforce one or another different policies).

In addition, a prototype MSIL (Microsoft Intermediate language) in-lined reference monitor (IRM) realization is now operational. It implements an aspect-oriented programming metaphor for MSIL assembly language (rather than for a high-level language). An aspect-oriented program comprises *aspects*, each of which consists of a *point-cut* and some *advice*. The point-cut is a predicate that specifies where to do rewriting in target code, and the advice specifies how to do the rewriting. Designing a point-cut language that provides complete visibility at a high-level into an assembly language is an interesting challenge; we plan next to turn our attention to this.

Cyclone Compiler

Today, our computing and communications infrastructure is built using unsafe, error-prone languages such as C or C++ where buffer overruns, format string errors, and space leaks are not only possible, but frighteningly common. In contrast, type-safe languages, such as Java, Scheme, and ML, ensure that such errors either cannot happen (through static type-checking and automatic memory management) or at least are caught at the point of failure (through dynamic type and bound checks.) This fail-stop guarantee is not a total solution, but it does isolate the effects of failures, facilitates

testing and determination of the true source of failures, and enables tools and methodologies for achieving greater levels of assurance.

The obvious question is: "Why don't we re-code our infrastructure using type-safe languages?" Though such a technical solution looks good on paper, the cost is simply too large. For instance, today's operating systems consist of tens of millions of lines of code. Throwing away all of that C code and reimplementing it in, say Java, is simply too expensive, and in some situations, not even possible.

An alternative approach is to try to adapt safe language technology to legacy C and C++ systems. The ideal solution should:

- catch most errors at compile time,
- give a fail-stop guarantee at run time, and
- scale to millions of lines of code

while simultaneously:

- minimizing the cost of porting the code from C/C++,
- interoperating with legacy code,
- giving programmers control over low-level details needed to build systems.

As a step towards these goals, we have been developing Cyclone, a type-safe programming language that can be roughly characterized as a "superset of a subset of C." The type system of Cyclone accepts many C functions without change and uses the same data representations and calling conventions as C for a given type constructor. It also rejects many C programs to ensure safety. For instance, it rejects programs that perform (potentially) unsafe casts, that use unions of incompatible types, that (might) fail to initialize a location before using it, that use certain forms of pointer arithmetic, or that attempt to do certain forms of memory management.

All of the analyses used by Cyclone are local (i.e., intra-procedural) so we can ensure scalability and separate compilation. The analyses have also been carefully constructed to avoid unsoundness in the presence of threads. The price paid is that programmers must sometimes change type definitions or prototypes of functions, and occasionally they must rewrite code.

We find that programmers must touch about 10% of the code when porting from C to Cyclone. Most of the changes involve choosing pointer representations and only a very few involve region annotations (around 0.6

% of the total changes.) In the future, we hope to minimize this burden by providing a porting tool that utilizes a more global analysis to determine the appropriate representation.

The performance overhead of the dynamic checks depends upon the application. For systems applications, such as a simple web server, we see no overhead at all. This is not surprising, as these applications tend to be I/O-bound. For scientific applications, we see a much larger overhead (around 5x for a naive port, and 3x with an experienced programmer). We believe much of this overhead is due to bounds and null pointer checks on array access. We have incorporated a simple, intra-procedural analysis to eliminate many of those checks and indeed, this results in a marked improvement. However, some of the overhead is also due to the use of "fat pointers" and the fact that GCC does not always optimize struct manipulation. By unboxing the structs into variables, we may find a marked improvement.

Secure Program Partitioning

Our *secure program partitioning* provides the means to ensure that data confidentiality and integrity are preserved in distributed systems that contain untrusted hosts and mutually distrusting principals. This problem is particularly relevant to information systems used by mutually distrusting organizations, such as the dynamic coalitions that arise in military settings.

In our approach, programs are automatically partitioned into communicating subprograms that run on the available, partially trusted hosts. The partitioning automatically extracts a secure communications protocol, and if any host is subverted, then only principals that have explicitly stated trust in that host need fear a violation of confidentiality. That is, for a given principal p, the partitioned program we create is robust against attacks on hosts not trusted by p. To protect data integrity, information and code are also replicated across the available hosts.

We have implemented these techniques in Jif/split, an extension to our publicly released Jif compiler that statically enforces information flow control, in conjunction with a distributed run-time system that securely executes partitioned programs while guarding against subverted or malicious hosts. New protocols had to be developed in order to permit secure transfer of control between one group of host replicas and another. And to understand the practicality of our approach, secure distributed systems have been implemented using Jif/split, including various secure auction protocols. Performance of the system is quite reasonable, despite the fine-grained program

partitioning.

Publications Supported under this Grant

(1) Allegra Angus and Dexter Kozen. Kleene algebra with tests and program schematology. Technical Report 2001-1844, Computer Science Department, Cornell University, July 2001.

(2) Adam Barth and Dexter Kozen. Equational verification of cache blocking in LU decomposition using Kleene algebra with tests. Technical Report 2002-1865, Computer Science Department, Cornell University, June 2002.

(3) James Cheney and Ralf Hinze. Poor man's generics and dynamics. *Haskell Workshop 2002.* To appear.

(4) Karl Crary, Stephanie Weirich, and Greg Morrisett. Intensional polymorphism in type erasure semantics. *Journal of Functional Programming.* To appear.

(5) Dan Grossman. Existential Types for Imperative Languages. Type checking systems code. *Eleventh European Symposium on Programming* (Grenoble, France, April 2002), Lecture Notes in Computer Science Volume 2305, 21–35.

(6) D. Grossman, G. Morrisett, T. Jim, M. Hicks, J. Cheney, and Y. Wang. Region-based memory management in Cyclone. *ACM Conference on Programming Language Design and Implementation* (Berlin, Germany, June 2002), 282–293.

(7) David Harel, Dexter Kozen, and Jerzy Tiuryn. Dynamic logic. In D.M. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, Volume 4, Kluwer, 2nd edition, 2002, 99–217.

(8) T. Jim, G. Morrisett, D. Grossman, M. Hicks, J. Cheney, and Y. Wang. Cyclone: A safe dialect of C. *Usenix Annual Technical Conference* (Monterey, CA, June 2002).

(9) Dexter Kozen. On two letters versus three. In Zoltán Ésik and Anna Ingólfsdóttir, editors, *Proceedings of Workshop on Fixed Points in Computer Science* (FICS'02), July 2002, 44–50.

(10) Dexter Kozen. Some results in dynamic model theory (abstract). In E. A. Boiten and B. Möller, editors, *Proceedings Conference on Mathematics of Program Construction* (MPC'02), Lecture Notes in Computer Science Volume 2386 (July 2002), 21.

(11) Dexter Kozen. On the complexity of reasoning in Kleene algebra. *Information and Computation.* To appear.

(12) Dexter Kozen. Computational inductive definability. Submitted for publication.

(13) Dexter Kozen and Matt Stillerman. Eager class initialization for Java. *Proceedings 7th International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems* (FTRTFT'02), IFIP, (Germany, Sept. 2002). To appear.

(14) Dexter Kozen and Jerzy Tiuryn. On the completeness of propositional Hoare logic. *Information Sciences 139*, 2001. 187–195.

(15) G. Morrisett. Type checking systems code. *Eleventh European Symposium on Programming* (Grenoble, France, April 2002), Lecture Notes in Computer Science Volume 2305, 1–5.

(16) G. Morrisett, K. Crary, N. Glew, and D. Walker. Stack-based typed assembly language. *Journal of Functional Programming 12*, No. 1 (January 2002), University Press, Cambridge, England, 43–88.

(17) Andrei Sabelfeld and Andrew C. Myers. End-to-end security via program analysis. Submitted for publication.

(18) Frederick Smith. *Certified Run-Time Code Generation.* Ph.D. Thesis, Cornell University, January 2002.

(19) Frederick Smith, Dan Grossman, Greg Morrisett, Luke Hornof, and Trevor Jim. Compiling for template-based run-time code generation. *Journal of Functional Programming*, Special issue on Semantics, Applications, and Implementation of Program Generation. To appear.

(20) Stephanie Weirich. Higher-order intensional type analysis. *Eleventh European Symposium on Programming* (Grenoble, France, April 2002), Lecture Notes in Computer Science Volume 2305, 98–114.

(21) Stephanie Weirich. *Programming With Types.* Ph.D. Thesis, Cornell University, July 2002.

(22) Steve Zdancewic and Andrew C. Myers. Secure information flow and linear continuations. *Higher Order and Symbolic Computation, 15* (2-3), 2002. To appear.

(23) Steve Zdancewic, Lantian Zheng, Nathaniel Nystrom, and Andrew C. Myers. Untrusted hosts and confidentiality: Secure program partitioning. *Proceedings of the 18th ACM Symposium on Operating Systems Principles* (Banff, Canada, October 2001), 1–14.

(24) Steve Zdancewic, Lantian Zheng, Nathaniel Nystrom, and Andrew C. Myers. Secure program partitioning. *ACM Transactions on Computing Systems.* To appear.

(25) Lantian Zheng, Stephen Chong, Steve Zdancewic, and Andrew C. Myers. Enforcing end-to-end integrity with replicated code partitions. Submitted for publication, May 2002.

DoD Interactions and Technology Transitions

- Schneider chaired a study for DARPA IPTO Program Manager Jay Lala on promising research directions for Self-Healing Networked Information Systems.

- Morrisett and Schneider continue working with Microsoft on developing a .NET version of our in-lined reference monitor (IRM) approach to security-policy enforcement. Next Fall, Cornell graduate student Kevin Hamlen will spend a semester visiting Microsoft Research (in Cambridge, England). The MSIL rewriter we have developed uses some software developed there, and this visit will speed the implementation of our new IRM tool.

- Researchers at Carnegie-Mellon University, Princeton University, University of California (Riverside), University of Newcastle-Upon-Tyne, and Intel Research are all now building on PoET/PSLang IRM tools developed by Schneider and collaborators.

- Further public releases of Myers' Jif compiler have been made available at the Jif web site, http://www.cs.cornell.edu/jif. The Jif language extends the Java programming language with support for information flow control. The Jif compiler is implemented on top of the Polyglot extensible compiler framework for Java. The Polyglot framework has also been released publicly at http://www.cs.cornell.edu/projects/polyglot,

and researchers at Princeton University are using this framework in their own research. The releases of both Jif and Polyglot are provided as Java source code and work on Unix and Windows platforms.

- AT&T research is working with us to develop the Cyclone language, compiler, and tools. In addition, researchers at the University of Maryland, the University of Utah, Princeton, and the University of Pennsylvania, and Cornell are all using Cyclone to develop research prototypes.